# 第4章 使用聚类算法结合原测试集提高压缩率

聚类算法是机器学习中涉及对数据进行分组的一种算法。在给定的数据集中，我们可以通过聚类算法将其分成一些不同的组。在理论上，相同的组的数据之间有相同的属性或者是特征，不同组数据之间的属性或者特征相差就会比较大。聚类算法是一种非监督学习算法，并且作为一种常用的数据分析算法在很多领域上得到应用。基于此，本文将聚类算法与数据压缩相结合，找到了一种提高压缩率的方法。第三章使用预填充测试集结合距离优先法则选取了基向量，最终获得了较好的压缩增益，但是基向量的选取始终是随机的，如果使用聚类算法将原测试集的列向量进行划分，以聚类中心作为基向量生成主分量集，在一定程度上能增大与原测试集的相似程度，提高残差集的压缩率。

## 4.1 聚类算法

### 4.1.1 **DBSCAN**聚类算法

DBSCAN（Density-Based Spatial Clustering of Applications with Noise，它是一种基于高密度连通区域的、基于密度的聚类算法，能够将具有足够高密度的区域划分为簇，并在具有噪声的数据中发现任意形状的簇，即要求在聚类空间中的一定区域内所包含对象(点或其他空间对象)的数目不小于某一给定阈值。过滤低密度区域，发现稠密度样本点。同一类别的样本，他们之间是紧密相连的，即在该类别任意样本周围不远处一定有同类别的样本存在。下面介绍DBSCAN算法的基本定义以及思路。

假设当前有假设样本集是

1. ϵ-邻域：对于，其ϵ-邻域包含样本集D中与的距离不大于ϵ的子样本集。
2. 核心对象：对于任一样本，如果其ϵ-邻域对应的子样本集个数至少包含MinPts个样本，则是核心对象。
3. 密度直达：如果位于的ϵ-邻域中，且是核心对象，则称由密度直达。
4. 密度可达：对于和,如果存在样本序列满足，, 且，由密度直达，则称由密度可达。也就是说，密度可达满足传递性。此时序列中的传递样本均为核心对象，因为只有核心对象才能使其他样本密度直达。
5. 密度相连：对于和，如果存在核心对象样本，使和均由密度可达，则称和密度相连。

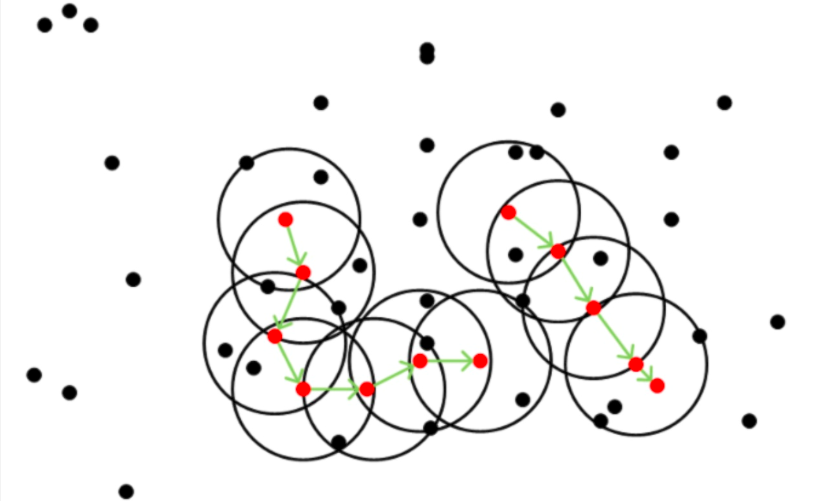


图4.1 DBSCAN图示

 从上图4.1可以很容易看出理解上述定义，图中MinPts=5，红色的点都是核心对象，因为其ϵ-邻域至少有5个样本。黑色的样本是非核心对象。所有核心对象密度直达的样本在以红色核心对象为中心的超球体内，如果不在超球体内，则不能密度直达。图中用绿色箭头连起来的核心对象组成了密度可达的样本序列。在这些密度可达的样本序列的ϵ-邻域内所有的样本相互都是密度相连的。

由密度可达关系导出的最大密度相连的样本集合，即为我们最终聚类的一个类别，或者说一个簇。这个DBSCAN的簇里面可以有一个或者多个核心对象。如果只有一个核心对象，则簇里其他的非核心对象样本都在这个核心对象的ϵ-邻域里;如果有多个核心对象，则簇里的任意一个核心对象的ϵ-邻域中一定有一个其他的核心对象，否则这两个核心对象无法密度可达。这些核心对象的ϵϵ-邻域里所有的样本的集合组成的一个DBSCAN聚类簇。

在此算法中有两个关键点，一个是ϵ 某一样本的邻域距离阈值（即领域半径）一个是MinPts 某一样本的距离为 ϵ 的邻域中样本个数的阈值（即最少点个数）。将其与数据压缩相结合，即ϵ 为列向量之间的欧几里得距离。原测试集中如果有大于等于MinPts个列向量与当前列向量之间的欧几里得距离小于ϵ，就可以确定当前列向量为一个中心点。根据电路的大小，需要选取的基向量数目为k，那么可以设计合理的ϵ 和MinPts确定聚类中心即可。

### 4.1.2 kmeans聚类算法

Kmeans算法又叫做k-平均算法（英文：k-means clustering）源于信号处理中的一种向量量化方法，现在则更多地作为一种聚类分析方法流行于数据挖掘领域。k-平均聚类的目的是：把 n个点划分到k个聚类中，使得每个点都属于离他最近的均值（此即聚类中心）对应的聚类，以之作为聚类的标准。

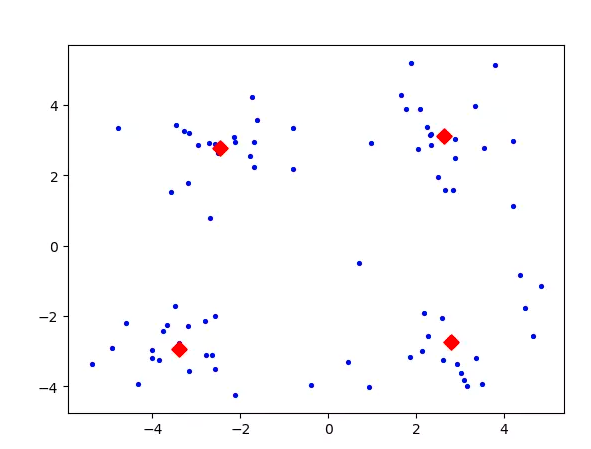
Means 是发现给定数据集的 K 个簇的聚类算法, 之所以称之为 K-均值 是因为它可以发现 K 个不同的簇, 且每个簇的中心采用簇中所含值的均值计算而成.簇个数 K 是用户指定的, 每一个簇通过其质心（centroid）, 即簇中所有点的中心来描述。聚类与分类算法的最大区别在于, 分类的目标类别已知, 而聚类的目标类别是未知的。

Kmeans中的专业术语如下所示：

簇：所有数据的点集合，簇中的对象是相似的。

质心：簇中所有点的中心（计算所有点的均值而来）.

SSE：Sum of Sqared Error（误差平方和）, 它被用来评估模型的好坏，SSE 值越小，表示越接近它们的质心. 聚类效果越好。由于对误差取了平方，因此更加注重那些远离中心的点（一般为边界点或离群点）。



从图像中的数据分布可以看出，样本点大致分为四簇。所以设置初始聚类中心 k=4，如果对数据不是很了解，可以先设置一个k，进行聚类之后，再根据结果的聚类效果来调整聚类中心的数量。上图可以看成二维坐标，其中每一个点都有相对应的x坐标和y坐标，通过欧几里得距离公式计算出SSE。

Kmeans算法的计算流程如下所示：

1. 首先确定一个k值，即我们希望将数据集经过聚类得到k个集合。一般而言k值并不是事先所知晓的需要尝试多次。
2. 从数据集中随机选择k个数据点作为质心。
3. 对数据集中每一个点，计算其与每一个质心的距离（如欧式距离），离哪个质心近，就划分到那个质心所属的集合。
4. 把所有数据归好集合后，一共有k个集合。然后重新计算每个集合的质心。
5. 如果新计算出来的质心和原来的质心之间的距离小于某一个设置的阈值（表示重新计算的质心的位置变化不大，趋于稳定，或者说收敛），便可以认为聚类已经达到期望的结果，算法终止。
6. 如果新质心和原质心距离变化很大，需要迭代3~5步骤。

通过上述的步骤基本上可以将一组数据划分成为k个聚类，如下我们将其转化为数学表达式，假设簇划分为,则我们的目标是最小化平方误差E：



其中为的均值向量，有时也称为质心，表达式为：



为了进一步理解kmeans算法，下面以二维坐标系为例，讲述如何划分聚类。

在二维坐标系中有6个点分别为p1(0,0)，p2(1,2 )，p3(3，1)，p4(8，8),

P5(9，10)，p6(10，7)，点与点之间的距离直接使用欧几里得距离公式计算，如果点并不是二维的坐标轴上，而是在三维、四维甚至n维，其距离计算公式为对应位两两相减求平方和之后开更根号。由于数据量较小，通过观察可得本数据可以划分为两个聚类，为了演示效果直接令k=2。初始时随机选两个点p1和p2，计算出剩余点与其距离如下表所示：

|  |  |  |
| --- | --- | --- |
|  | P1 | P2 |
| P3 | 3.16 | 2.24 |
| P4 | 11.3 | 9.22 |
| P5 | 13.5 | 11.3 |
| P6 | 12.3 | 10.3 |

按照距离为原则划分两组中的数据为：组A中只有p1，组B中为其他点。

第二步分别计算A组和B组的质心，A组质心为p1=(0,0),B组质心为pB=(6.2，5.6)，质心的计算公式为簇中各个向量相加取平均值即可。第三步，以质心为原点计算每一个点到质心的距离再次分组距离分布如下所示：

|  |  |  |
| --- | --- | --- |
|  | P1 | pB |
| P2 | 2.24 | 6.3246 |
| P3 | 3.16 | 5.6036 |
| P4 | 11.3 | 3 |
| P5 | 13.5 | 5.2154 |
| P6 | 12.2 | 4.0497 |

第二次分组结果为组A：p1、p2、p3。组B：p4、p5、p6。再次计算质心轮询上面的步骤直至组A和组B中的点不发生变化，迭代结束。

根据上述实验思路给出具体的算法流程伪代码如下所示：

|  |
| --- |
| 算法4.1 kmeans  输入：类簇个数K，迭代终止阈值 输出：聚类结果 |
| For k = 0,...,K  =rand()%COUNT // 初始化K个类簇的聚类中心  For t=1，2，3,...,T  For every  // 所有数据对象  Dis(,) // 将归于某一个聚簇中  End for  For k = 0,....,k  =avg(cluster(k)) // 将每一个聚簇的平均值作为新的聚类中心  Diff = new(cluster)-old(cluster) // 计算出新聚簇与老聚簇之间的差距  If Diff <  Then return res // 如果新旧聚簇无明显变化则终止循环  Break；  End if  End for |

Kmeans算法其优势是简单且易于实现，收敛速度快，对于结果密集型且区别明显的聚类簇分类效果好，但是其缺陷也非常明显，首先我们需要手动确定聚类数K，若原数据集并不能直接分为k个聚类有2k或者数据集较为分散没有明显特征，kmeans算法效果就会较差。其次kmeans对初始质心的选取依赖较大，不同的随机种子对结果影响非常明显。

基于此有学者提出了kmeans++算法，其改进主要针对Kmeans算法初始质心的随机选取。kmeans++在随机选取初始质心之前还进行了几个步骤（默认需要选取k个初始质心）。第一：从数据集中随机选取一个初始质心。第二：对于每个点，都计算其和最近的一个质心的距离D(x)并保存在一个数组里，然后把这些距离加起来得到Sum(D(x))。第三：取一个随机值，用权重的方式来取计算下一个质心。算法的实现为，先取一个能落在Sum(D(x))中的随机值Random，然后用Random -= D(x)，直到其<=0，此时的点就是下一个质心。

第四：重复第(2)和第(3)步直到所有的K个种子点都被选出来。接下来按照kmeans算法的步骤执行即可。

## 4.2 聚类算法结合测试集选取基向量

### 4.2.1 聚类算法的选择

在实验的过程中本人尝试了多种聚类算法，包括基于马尔可夫模型的聚类算法、均值漂移聚类算法、 凝聚层次聚类等，最终成功与实验相结合的只有上文提及的DBSCAN聚类算法和kmeans聚类算法。在本实验中，聚类对象为一个个向量，一个向量至少有100多位，距离的计算使用多维欧几里得距离。DBSCAN聚类算法虽然可以自动确定聚类个数，但是需要设置ϵ-邻域以及MinPts。本人经过多次尝试发现对于不同的电路，其ϵ-邻域以及MinPts均不相同，无法给出统一的标准，而且这两个值需要不断尝试才能获取较好的压缩增益，如果设置不当与随机选取无异。

Kmeans算法主要的缺陷有两点，第一无法确定数据集需要聚类的数目K，第二，只能随机选取初始质心。争对这两个缺陷本文给出了解决方案，首先本文根据电路的行来确定需要聚类的数目K，这样做有两点好处，第一对于不同的电路都适用，具有通用性。第二，会保证最终的聚类数不会过多，本实验最终要选择聚类中心充当基向量，若聚类数过多会导致基向量过多，增加硬件开销。

针对第二个缺陷，本文结合kmeans++的思想，使用一种极大化质心(基向量)距离的选取方式，保证了初始质心(基向量)之间的距离足够远。通过上述两种解决方案，本实验使用kmeans++算法对已填充的原测试进行聚类。下文将介绍详细过程。

### 4.2.2 基向量的选取

基向量选取完成后，测试集相对应的主分量集也随之确定。本章主要根据kmeans++算法给出基向量相应的计算方法。

在初始基向量的选取过程中，需要进行测试集预填充，是第三章完成的工作。由于本人采取的压缩方法是拆分压缩，将原测试集中的无关位填充为0即可。下面是选取的具体步骤：

（1）从我们生成的“填充测试集”随机选择一个列向量作为第一个聚类中心。

（2）对于数据集中的每一个列向量x，计算它与最近聚类中心(指已选择的聚类中心)的距离D(x)。

（3）在原测试集中选择一个新的列向量作为下一个聚类中心，将其类比为数学公式为，其中表示测试集中第一个列向量与基向量之间的最小距离。同理表示测试集中第N个列向量与基向量之间的最小距离，这个步骤主要保证基向量之间的距离最大。

（4）重复2和3直到k个聚类中心被选择出来，其中k的数量由我们原测试集的数据规模决定。

（5）选取出来k个初始的聚类中心之后，计算每个列向量到聚类中心的距离，这里采用欧式距离计算。

（6）每个列向量均能计算出k个距离，我们将其划分在最小距离所对应的聚类中心的类中。

（7）对应类簇中所有数据对象的均值，即为更新后该类簇中心。

（8）判断中心点是否满足迭代条件，如果不满足则返回第二步重复计算。迭代条件可以设定为迭代之后类族的中心点不发生变化或者直接初始化迭代的次数。

### 4.2.3 主分量集的获取

得到K个基向量后，将其依次进行相互异或，可得到一个列向量数为的矩阵，然后将这个矩阵取反且与原矩阵拼成一个新的矩阵，计算测试集中的列与新矩阵中的每一列的汉明距离，选取汉明距离最小时所对应新矩阵中的列作为主分量，构成主分量集，得到主分量集后，将它与原测试集进行异或，生成残分量集，最后对残分量集进一步压缩。主分量生成的具体过程如算法4.2所示。

|  |
| --- |
| 算法4.2 PrincipalComponentSetGenerationl (T)  输入：初始K个基向量 输出：主分量集 |
| /\*  N:Number of vectors  M:Number of columns  J[1:N,1 ceiling(log2N)]:Base vector matrix  T[1:N,1:M]:Test set size  \*/  Z=Xor(J) //将基向量两两间进行异或得到矩阵Z  Z=[Z,-Z] //将矩阵Z取反与原来的Z组成新矩阵  For i from 1 to M  d= Hamming(Z,T[1:N, i] //计算矩阵Z与原测试集中第i列的汉明距离  t= min(d ) //求出汉明距离d中绝对值最小所对应的索引k  Principal=Z[t] //提取主分量  PrincipalComponentSet add(Principal)  Return PrincipalComponentSet //返回主分量集  End for |

例如测试集A的基向量为**β**1(1,0,1,0,1)、**β**2(1,1,0,0,1)、**β**3(0,1,1,1,1)组成，通过基向量两两异或可生成矩阵Z，基向量为三列，对应的Z为7列。

然后将Z取反与原矩阵拼接成为全新矩阵，取反即为0和1位置互换。



最终计算出测试集A中的每一列与矩阵每一列的汉明距离dis，将dis所对应中的列向量作为当前测试集中相应列的主分量。例如测试集中第二列**β**2(1,1,0,0,1)与矩阵中第二列(1,1,0,0,1)的汉明距离为0，则原测试集第二列的主分量为(1,1,0,0,1)。以此类推，测试集每一列使用此运算将得到主分量集合。最终将主分量集合原测试集进行异或，计算出相应残差集，并运用编码压缩得到压缩率。

## 4.3实验结果

本实验使用kmeans++算法需要预先设定聚类数K，这在一定程度上影响了实验准确性。针对这一问题。实验结果分为两部分，第一部分根据电路大小选取基向量个数，并取得压缩率，同时与其他拆分压缩所取得的压缩率进行对比。第二部分，由于基向量需要存储代价，基向量的选取不能无限制的增加，本人在一定范围内动态增加或者减少基向量个数，观察压缩率的变化情况，大致绘制出基向量-压缩率之间的趋势变化图。

### 4.3.1 根据电路大小确定基向量数

根据电路大小选取基向量，其选取方式选取为其中N为测试集的行数。为了验证聚类算法的有效性，我们对 ISCAS’89中大部分电路进行了实验，包括S5378、S9234、S13207、S15850、S38417、S38584等电路，本章将挑选其中的电路做具体描述，并与其他方式所得压缩率做一个对比。

实验结果如下表4.1、4.2、4.3、4.4、4.5、4.6所示，本文选取了FDR、EFDR、ALT-FDR、RL-Huff、VIHC以及Golomb六种编码方式对变换拆分之后的残差集进行压缩，同时与使用哈达码变换达、直接预填充方式获取的压缩率进行对比。

第一列为电路名称，第二列为测试集直接压缩之后的结果，第三列是对测试集哈达码变换拆分之后的压缩率，第四列为先对测试集预填充拆分压缩之后的压缩率，第五列是由kmeans++聚类算法结合拆分压缩所的压缩率。

表 4.2 是 FDR 编码在各种情况下的压缩率比较，FDR 编码是一种单游程编码方式，其码表中对连续0子串进行编码，遇到1比特位直接使用00进行编码，当0串越长所对应的码字相对于原串越短，压缩率越高。由表中的数据可知，采用本章所提出方法计算的压缩率比测试集直接编码的平均高20%，比对原测试集进行哈达码变换相比，平均压缩率高百分之2.4，比预填充压压缩率平均高1.72%。

表4.2 FDR编码压缩率(%)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 电路 | 直接编码 | 哈达码变换 | 预填充 | 本方法 |
| s5378 | 47.98 | 67.51 | 68.43 | 70.76 |
| s9234 | 43.61 | 66.19 | 66.39 | 69.59 |
| s13207 | 81.31 | 89.65 | 91.69 | 92.29 |
| s15850 | 66.21 | 80.66 | 81.88 | 81.75 |
| s38417 | 43.27 | 72.44 | 73.04 | 75.35 |
| s38584 | 60.93 | 75.99 | 75.09 | 77.03 |
| 平均 | 57.22 | 75.41 | 76.08 | 77.80 |

表 4.3 是 EFDR 编码在不同情况下压缩率的比较，EFDR 编码是一种双游程编码方式，属于对 FDR 编码的一种扩展，既可以对0游程进行编码，也可以对1进行编码，对测试集中0和1的数目并没有具体的要求，只需要保证跳变数最少即可，如果测试集中0和1相继出现，压缩效果就会比较差，由表可知，采用本章所提出方法的压缩率比对测试集直接编码的压缩率平均高 12.37%，比哈达码变换的压缩率平均高 2.37%，比在预填充方法的压缩率平均高 1.58%。

表4.3 EFDR编码压缩率(%)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 电路 | 直接编码 | 哈达码变换 | 预填充 | 本方法 |
| s5378 | 53.67 | 64.50 | 65.60 | 67.75 |
| s9234 | 48.66 | 62.74 | 62.71 | 66.14 |
| s13207 | 82.49 | 88.89 | 90.95 | 91.60 |
| s15850 | 68.66 | 78.67 | 78.88 | 79.84 |
| s38417 | 62.02 | 71.63 | 71.71 | 74.06 |
| s38584 | 64.28 | 73.45 | 74.69 | 74.60 |
| 平均 | 63.30 | 73.30 | 74.09 | 75.67 |

表 4.4、4.5 分别是是 ALT-FDR 编码和 RL-Huff 编码在各种情况下的压缩率比较，ALT-FDR 编码和 RL-Huff 编码都是双游程编码，是根据跳变数最少来进行编码的。由表可知，采用本章所提出方法 ALT-FDR 编码和 RL-Huff 编码的压缩率分别比对测试集直接编码的压缩率平均高 12.42%和 13.73%，比哈达码变换的压缩率分别平均高 3.24%和4.45%，比预填充策所达到的平均压缩率分别高 1.7%和 1.98%。虽然上述两种编码方式属于双游程编码，但是较其他压缩方法获得的相对压缩增益是最高的，

表4.4 ALT-FDR编码压缩率(%)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 电路 | 直接编码 | 哈达码变换 | 预填充 | 本方法 |
| s5378 | 49.95 | 61.62 | 63.54 | 65.17 |
| s9234 | 44.96 | 58.31 | 58.89 | 62.72 |
| s13207 | 80.23 | 86.52 | 90.16 | 90.88 |
| s15850 | 65.83 | 75.76 | 76.78 | 77.82 |
| s38417 | 60.55 | 68.40 | 68.13 | 71.23 |
| s38584 | 61.13 | 69.70 | 72.09 | 71.97 |
| 平均 | 60.58 | 70.06 | 71.60 | 73.30 |

表4.5 RL-Huff编码压缩率(%)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 电路 | 直接编码 | 哈达码变换 | 预填充 | 本方法 |
| s5378 | 52.58 | 64.02 | 66.73 | 68.59 |
| s9234 | 47.26 | 60.33 | 63.16 | 67.07 |
| s13207 | 82.49 | 88.71 | 92.23 | 92.82 |
| s15850 | 67.35 | 77.33 | 79.47 | 80.57 |
| s38417 | 63.32 | 69.66 | 71.43 | 74.02 |
| s38584 | 62.40 | 71.05 | 72.91 | 74.74 |
| 平均 | 62.57 | 71.85 | 74.32 | 76.30 |

表 4.6 是 VIHC 编码在不同情况下压缩率的比较，VIHC 编码是一种单游程编码方式，采用“1”最少的原则进行拆分。由下表可知，采用本章所提出方法的压缩率比对测试集直接编码的压缩率平均高 19.25%，比哈达码变换的压缩率平均高2.39%，比预填充获取的平均压缩率高 0.06%，用 VIHC 编码也使压缩率平均达到了1.42%。

表4.6 VIHC编码压缩率(%)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 电路 | 直接编码 | 哈达码变换 | 最大相容 | 本方法 |
| s5378 | 51.75 | 69.63 | 70.78 | 72.81 |
| s9234 | 47.23 | 69.58 | 70.12 | 73.25 |
| s13207 | 83.55 | 92.20 | 93.74 | 94.20 |
| s15850 | 67.97 | 82.96 | 83.30 | 84.28 |
| s38417 | 53.39 | 74.79 | 75.74 | 77.86 |
| s38584 | 62.30 | 78.11 | 79.50 | 79.25 |
| 平均 | 61.03 | 77.89 | 78.86 | 80.28 |

从上表中的结果可知，本方法与直接编码、哈达码变换、从预填充这三种方法相比，平均压缩率都要高。

为了验证本方法确实有效，本文除了对ISCAS’89中大部分电路进行了实验，还对b15、b17、b20、b21等大电路进行了相关实验。大电路中虽然拥有较多的无关位，但是电路所包含的比特位比较多有些甚至上百万位，上文实验中提及的电路大部分都是10万位以内。实验结果如下表4.7、4.8、4.9、4.10所示，第一列为电路名称，第二列为当前电路的行数和列数，第三列是对测试集直接编码的压缩率，第四列为本方法所达到的压缩率。

下表为各个电路FDR编码下的的压缩率，可以看出大电路中本身就存在较多的无关位，直接压缩也可取得较高放入压缩增益，但是使用本方法比直接编码的压缩率平均提高近6%

表4.7 FDR编码压缩率(%)

|  |  |  |  |
| --- | --- | --- | --- |
| 电路 | 电路规模 | 直接编码 | 本方法 |
| b15 | 1630\*483 | 83.26 | 91.64 |
| b17 | 2555\*1452 | 89.93 | 94.10 |
| b20 | 5510\*522 | 84.15 | 90.54 |
| b21 | 5496\*522 | 84.13 | 90.63 |
| b22 | 3369\*767 | 85.10 | 89.94 |
| 平均 |  | 85.31 | 91.37 |

下表4.9为各个大电路在EFDR、VIHC、ALT-FDR和RL-Huff下的压缩率，由表中数据可得当前压缩方法确实有利于提高压缩率，即使电路规模相对较大，也能获得较好的压缩增益。

表3.7 其他编码方法的压缩率

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 电路 | 电路  规模 | 直接压缩 | | | | 本方法 | | | |
| EFDR VIHC AFDR RL-Huff | | | | EFDR VIHC AFDR RL-Huff | | | |
| b15 | 1630\*483 | 85.99 | 86.72 | 85.12 | 87.01 | 91.08 | 92.91 | 90.61 | 92.04 |
| b17 | 2555\*1452 | 91.70 | 91.88 | 91.10 | 92.30 | 93.81 | 94.90 | 93.56 | 94.46 |
| b20 | 5510\*522 | 86.56 | 86.44 | 85.81 | 87.50 | 89.92 | 91.96 | 89.51 | 91.07 |
| b21 | 5496\*522 | 86.74 | 86.45 | 86.04 | 87.69 | 89.98 | 92.09 | 89.55 | 91.16 |
| b22 | 3369\*767 | 86.77 | 86.90 | 86.00 | 87.30 | 89.35 | 90.99 | 88.80 | 90.20 |
| 平均 |  | 87.55 | 87.68 | 86.81 | 88.36 | 90.83 | 92.57 | 90.41 | 91.79 |

### 4.3.2 动态选取基向量数

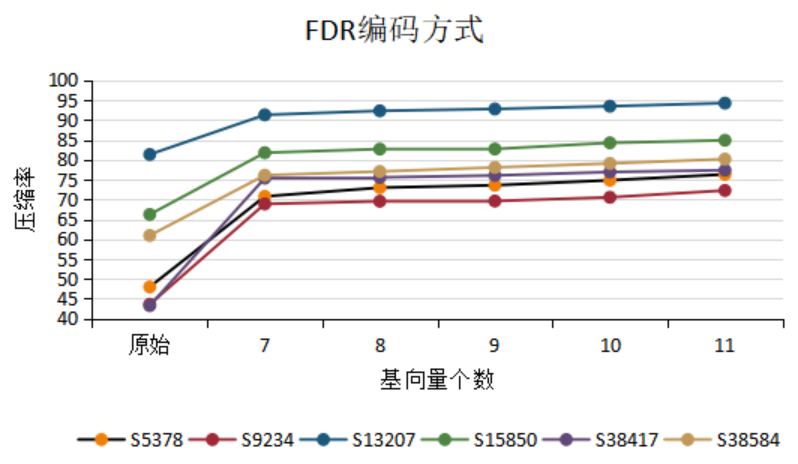
使用kmeans++算法并不能自动确定电路的聚类数，需要事先给定。虽然上文通过电路行数选取基向量获得了较好的压缩增益，并且在同等条件下比其他压缩方法取得的压缩率也更高，但并不能证明按照电路行数来选取聚类数会获得最高的压缩增益。基于此，本人在研究过程中，动态地选取了基向量数，进一步计算出确定基向量数与取得最终压缩率之间的关系。

本实验依然对ISCAS’89中大部分电路进行了实验，包括S5378、S9234、S13207、S15850、S38417、S38584电路，下图是当前算法在FDR编码方式下，压缩率的变化情况，第一列为电路名，第二列为原始压缩率，第三列到第七列为选取7到11个聚类数电路所能达到的压缩率。

表4.8 FDR编码压缩率(%)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 电路 | 直接编码 | 7列 | 8列 | 9列 | 10列 | 11列 |
| s5378 | 47.98 | 70.76 | 72.96 | 73.57 | 74.82 | 76.3 |
| s9234 | 43.61 | 68.86 | 69.54 | 69.59 | 70.54 | 72.25 |
| s13207 | 81.31 | 91.28 | 92.29 | 92.76 | 93.45 | 94.23 |
| s15850 | 66.21 | 81.75 | 82.67 | 82.69 | 84.24 | 84.88 |
| s38417 | 43.27 | 75.35 | 77.54 | 76.03 | 76.9 | 77.36 |
| s38584 | 60.93 | 76.09 | 77.03 | 78.05 | 79.05 | 80.15 |
| 平均 | 57.22 | 77.35 | 78.67 | 78.78 | 79.83 | 80.86 |

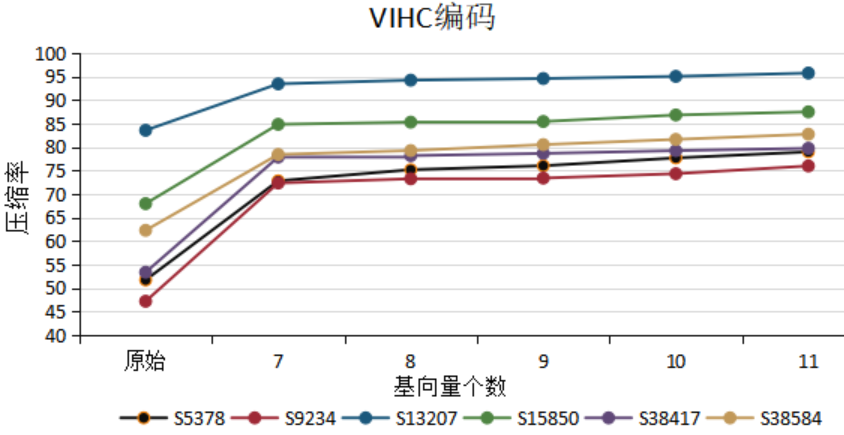
根据上表我们绘制出如下的折线图4.1，从下图可以看出随着基向量数的增加，压缩率整体呈上升趋势，整体来看当基向量数从7列增加至8列时压缩率提高了1.32%，之后随基向量增加压缩增益的增加幅度稍微有所降低。



下表4.10、图4.2是当前算法在VIHC编码方式下，压缩率的变化情况。

表4.10 VIHC编码压缩率(%)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 电路 | 直接编码 | 7列 | 8列 | 9列 | 10列 | 11列 |
| s5378 | 51.75 | 72.81 | 75.18 | 76.02 | 77.68 | 78.97 |
| s9234 | 47.23 | 72.35 | 73.25 | 73.4 | 74.33 | 75.98 |
| s13207 | 83.55 | 93.42 | 94.2 | 94.55 | 95.01 | 95.71 |
| s15850 | 67.97 | 84.82 | 85.28 | 85.42 | 86.8 | 87.47 |
| s38417 | 53.39 | 77.86 | 78.17 | 78.64 | 79.23 | 79.73 |
| s38584 | 62.30 | 78.42 | 79.25 | 80.51 | 81.62 | 82.72 |
| 平均 | 61.03 | 79.95 | 80.89 | 81.42 | 82.445 | 83.43 |



下表4.11、4.12以及图4.3、4.4分别为当前算法在RL\_Huff、AFDR编码下压缩率的变化情况。

表4.11 RL\_Huff编码压缩率(%)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 电路 | 直接编码 | 7列 | 8列 | 9列 | 10列 | 11列 |
| s5378 | 68.59 | 71.26 | 71.79 | 73.59 | 74.91 | 68.59 |
| s9234 | 66.05 | 67.07 | 67.05 | 68.17 | 70.18 | 66.05 |
| s13207 | 91.81 | 92.82 | 93.23 | 93.79 | 94.62 | 91.81 |
| s15850 | 80.57 | 81.73 | 81.77 | 83.65 | 84.34 | 80.57 |
| s38417 | 74.02 | 74.28 | 74.71 | 75.57 | 76.17 | 74.02 |
| s38584 | 73.7 | 74.74 | 76.13 | 77.42 | 78.78 | 73.7 |
| 平均 | 75.79 | 76.98 | 77.45 | 78.70 | 79.83 | 75.79 |

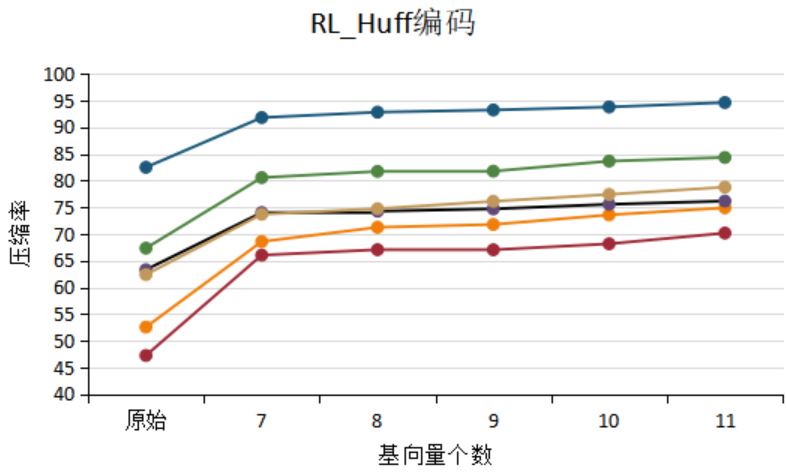
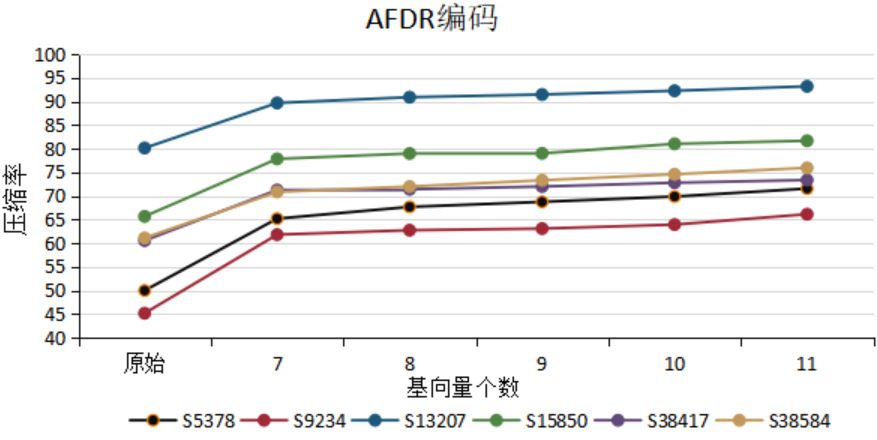


表4.12 AFDR编码压缩率(%)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 电路 | 直接编码 | 7列 | 8列 | 9列 | 10列 | 11列 |
| s5378 | 49.95 | 65.17 | 67.66 | 68.72 | 69.85 | 71.53 |
| s9234 | 45.14 | 61.78 | 62.72 | 63.06 | 63.91 | 66.11 |
| s13207 | 80.12 | 89.66 | 90.88 | 91.45 | 92.24 | 93.18 |
| s15850 | 65.64 | 77.82 | 78.96 | 78.99 | 81.01 | 81.66 |
| s38417 | 60.52 | 71.23 | 71.38 | 71.97 | 72.77 | 73.34 |
| s38584 | 61.09 | 70.82 | 71.97 | 73.29 | 74.57 | 75.93 |
| 平均 | 60.41 | 72.75 | 73.93 | 74.58 | 75.73 | 76.96 |



从上述的图表可以看出，随着基向量个数的增加，压缩率呈上升趋势，仔细观察可以发现，当基向量增加一个，压缩率提升百分之1左右。总而言之当前压缩方法无论是根据电路大小确定基向量，还是动态选取基向量均能取得不错的压缩效果。

## 4.4 小结

本章提出了一种使用聚类算法结合原测试生成主分量的数据压缩方法，该方法先通过预填充方法，去除原测试集中的无关位，然后根据电路大小确定基向量数。由于初始基向量的个数会影响最终的压缩率，并且无法确定初始聚类数，本人进行了动态选取基向量数相关实验，同时也使用当前压缩方法对大电路进行了测试。实验结果表明，使用kmeans++聚类算法集合原测试的压缩方法能大大提高压缩率。